5585P7148 *Patent*

UNITED STATES PATENT APPLICATION

for

DIRTY DATA PROTECTION FOR CACHE MEMORIES

Applicant(s):

Peter L. Fu

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard
Los Angeles, CA 90026-1026
(303) 740-1980

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EV064118311US
Date of Deposit February 7 2002
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Debbie Peloquin
(Typed or printed name of person mailing paper or fee)

Debbie Peloquin
(Signature of person mailing paper or fee)

DIRTY DATA PROTECTION FOR CACHE MEMORIES

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0002] This invention relates to the field of cache memories, and to a method for protecting dirty data in cache memories.

BACKGROUND OF THE INVENTION

[0003] Computer memories take many different forms. The slowest memories, and usually the farthest from the central processing unit (CPU), are virtual memories, such as a disk drive. The fastest, and usually the nearest, memories, include main memory, registers, and caches. Of these, main memory is the slowest. Since caches are closer to the CPU, they allow information, such as recently used instructions and/or data, to be quickly accessed by the CPU. However, since caches are relatively expensive, and limited in size, the data that can be stored on a cache memory is limited.

[0004] Cache memories can be used to store read-only data and read/write data. When read-only data is cached, data that is read from a memory can be stored in a cache so that the next time the data needs to be read, it can be read from the faster cache rather than the slower system memory. An example of read-only data that is cache read from memory is instructions to a program. Read-only data is not prone to data loss since any data that gets corrupted in the cache can always be read from memory.

[0005] When read/write data is cached, data that is to be written to system memory can be written to the cache instead. The way that a cache handles writes is called the "write policy" of the cache. There are different write policies, two of which are described below.

[0006] A cache can be a write-back cache. When a write is made to system memory at a location that is currently cached, the new data is only written

to the cache, and not to the system memory. Correspondingly, if another memory location needs to use the cache line where this data is stored, the currently stored data needs to be saved - i.e., written back - to the system memory so that the line can be used by the new memory location.

[0007] A cache can, alternatively, be a write-through cache. With this method, everytime a processor writes to a cached memory location, both the cache and the underlying memory location are updated. Cache written data may include any data, for example, a user-modified document.

[0008] While the write-through policy is more recoverable than the write back policy with respect to recovering from cache corruption, the write-back policy provides better performance at the risk of memory integrity, and are, therefore, prone to data loss due to cache errors.

[0009] In particular, where the write-back policy is utilized (i.e., new data that is to be written to a memory location that is cached is written to the cache and not to the memory location, and the new data is saved - written back to memory - when the cache line needs to be used by another memory location, for example), there is the risk that the cache may be corrupted before new data is saved.

[0010] Data in the cache which matches the data in a corresponding memory location is called clean data; and data in the cache which does not match the data in a corresponding memory location is called dirty data.

SUMMARY OF THE INVENTION

[0011] In one aspect of the invention, a method for maintaining dirty data is disclosed. This method comprises receiving a request to write data to a memory location that is cached, and then actually writing that data to a plurality of cache lines and marking those cache lines as dirty lines.

[0012] In another aspect of the invention, a method for saving, or writing back the dirty data to memory is disclosed. The method comprises reading dirty data from a cache line, and determining if the dirty data is corrupt. If it is not corrupt, then the line is marked available, and all dirty data lines corresponding to the read dirty data line are marked invalid.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0014] FIG. 1 is a block diagram illustrating a cache line in accordance with embodiments of the invention.

[0015] FIG. 2 is a block diagram illustrating a memory and cache system in accordance with embodiments of the invention.

[0016] FIG. 3 is a flowchart illustrating a method for caching data in accordance with embodiments of the invention.

[0017] FIG. 4 is a flowchart illustrating a method for writing back selected dirty lines in accordance with embodiments of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0018] In one aspect of the invention, a cost-efficient method for protecting dirty data in cache memories is provided. When an instruction to write data to a memory location is received, and that memory location is a cached memory location, the data is instead written to a plurality of associated cache lines. When data is written back to memory, one of the associated cache lines is read. If the cache line is not corrupt, it is written back to the appropriate memory location and marked as clean. In one embodiment, if associated cache lines exist, they are invalidated. In another embodiment, the other associated cache lines may be read for the highest confidence of reliability, and then invalidated.

[0019] The present invention includes various operations, which will be described below. The operations of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the operations. Alternatively, the operations may be performed by a combination of hardware and software.

[0020] The present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks,

CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs (Read Only Memories), RAMs (Random Access Memories), EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electromagnetic Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions.

[0021] Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Accordingly, herein, a carrier wave shall be regarded as comprising a machine-readable medium.

Introduction

[0022] Since caches are highly volatile and prone to error, they typically have error detection capabilities. For example, to detect the corruption of data, parity checking may be used. In parity checking, a parity bit is used to check for errors in groups of transferred data bits. In parity checking, the number of ones in each successfully transmitted set of bits must be an odd or even number (depending on the implementation). Failure of a parity check results in an error indicating that the transmitted data is corrupt.

[0023] When data in a cache becomes corrupted, the data can be recovered using error correction or replacement capabilities. For example,

Hamming codes can be used to correct errors, and duplication may be used to replace errors. While Hamming codes can be used to correct errors, they can become costly as the number of bits to correct increases. Likewise, while duplication can be a reliable method of data recovery, it, too, can be a very costly mechanism since it entails storing multiple copies of all data. For example, if one bit is stored in memory, two bits of the same value are stored in the cache. If an error is detected in one copy, the copy is thrown away, and the other copy may be used.

[0024] Another mechanism that can be used is similar to a redundant array of individual disks (RAID) approach. This mechanism is a disk drive arrangement that uses a parity drive. Thus, if there are n drives, the n+1 drive is a parity of the other n. Analogously, a cache memory can be segregated into n slices, and the RAID approach would require an n+1 slice to be a parity of the other n slices. Like data duplication, this method of data recovery is a very costly mechanism, in addition to being logically complex.

[0025] In embodiments of the invention, dirty data duplication is used to duplicate dirty cache lines only, as opposed to duplication of all cache lines. Dirty data duplication is effective under the presumption that the chances of multiple copies of the dirty data becoming corrupt in the cache are much lower than the chances of only a single copy becoming corrupt in the cache.

Caching

[0026] When an instruction to write data to a cached memory location is received, the data is written to a plurality of associated cache lines. In described embodiments, a four way set-associative cache is used whereby a single memory location may be indexed to any one of four cache lines in the cache. Furthermore, in described embodiments, data is duplicated to two cache lines.

[0027] However, the invention is not limited to any particular number of duplicated cache lines, except that the number of cache lines is greater than one. Moreover, the invention is not limited to a four way set-associative cache, nor is it limited to a set-associative cache. For instance, embodiments of the invention may be described with respect to a two way set-associative cache.

Illustrative Cache Line

[0028] As illustrated in FIG. 1, a cache line C*x* 100, where *x* is a descriptive cache line number, comprises a valid bit 102, state bits 104, a tag address 106, data 108, and checkbits 110. The valid bit is used to determine the content of the data. If the data is valid, it may be clean data or dirty data, and if it is invalid, it may be used to store data. Since clean data matches data in the corresponding memory location, and thus may be overwritten (while dirty data should never be overwritten since it is a different and more recent copy than what is stored in main memory), a valid bit indicates that the data contained in the cache line should at least be considered, whereas an invalid bit indicates that the line can be used without consideration. As used herein, both clean cache lines and invalid cache lines are considered to be available cache lines.

[0029] The state bits are used to determine the type of data stored in the cache line. For instance, the state of the data may be clean or dirty. While other states may exist, they are omitted from discussion herein so as to not obscure the invention at hand. The tag address comprises the memory location to which the data corresponds, and the checkbits comprise bits for error detection. If the checkbits detect an error, then the data is said to be corrupt; otherwise, the data is uncorrupt.

Associated Set and Duplicates

[0030] An associated set is a set of *n* cache lines, where *n* is the number corresponding to an *n*-way set-associative cache. As each cached memory location corresponds to any one of *n* lines in an *n*-way set-associative cache, the *n* lines are considered to be part of an associated set. Thus, in described embodiments where four way set-associative caches are illustrated, a given memory location corresponds to an associated set comprising four cache lines. Cache lines within an associated set are associated cache lines, and cache lines corresponding to the same memory location are duplicates.

Available Cache Lines

[0031] Available cache lines comprise those cache lines that are marked clean or invalid. This can be determined, for example, by examining the valid bit in the cache line to determine whether or not it is set, and within the valid bit, what the state of the data is. If no available cache lines exist, then write-back to memory occurs in accordance with a replacement policy to free up cache lines.

Caching Example

[0032] In a four way set-associative cache, all cache lines are associated with one another such that a memory location can map to any one of the four lines in the set, and can use any of the four lines in the set for duplication, assuming they are available.

[0033] A caching example in accordance with embodiments of the invention is illustrated in FIG. 2, where a memory 200 and a corresponding four way set associative cache 202 is illustrated. In this example, C_1 214, C_2 216, C_3 218, and C_4 220 are associated with one another in an associated set 230 (i.e., a given memory location can map to any one of C_1 214, C_2 216, C_3 218, and C_4 220), and C_5 222, C_6 224, C_7 226, and C_8 228 are associated with one another in an associated set 232. Within these sets, the following are some possible combinations:

[0034] • C_1 214 is a duplicate of C_2 216, and C_3 218 is a duplicate of C_4 220.

[0035] • C_1 214 and C_2 216 are clean lines, and C_3 218 is a duplicate of C_4 220.

[0036] • C_1 214, C_2 216 C_3 218, and C_4 220 are all clean lines.

[0037] • C_5 222 is a duplicate of C_6 224, and C_7 226 and C_8 228 are clean lines.

[0038] • C_5 222 and C_6 224 are clean lines, and C_7 226 is a duplicate of C_8 228.

[0039] • C_5 222, C_6 224, C_7 226, C_8 228 are all clean lines.

[0040] Of course, other combinations are possible, and are dynamic since the state of the lines (i.e., clean or dirty) may change during the course of cache reads and writes.

[0041] As an example, an instruction to write data to a memory 200 at a memory location that is cached 204, 206, 208, 210, 212 is received. Instead of writing the data to the memory location, it is written to two available cache lines 214-228 in an associated set 230, 232 of the corresponding cache 202. The two available cache lines in the set then become duplicates.

[0042] For example, if the core processor (as opposed to a general processor that typically includes the core processor and memory) receives an instruction to write data to memory location 206, and memory location 206 indexes to cache lines in associated set 230, then the a cache controller could write to any two available cache lines in the associated set 230. Thus, if cache lines 214 and 216 are available, then the controller would write to those two cache lines. Furthermore, if no cache lines are available, the cache controller would invoke a replacement policy (to be discussed) to free up cache lines.

[0043] FIG. 3 is a flowchart illustrating a method in accordance with FIG. 2. It begins at block 300 and continues to block 302 where an instruction to write data to a cached memory location is received. At block 304, an associated set is

determined. At block 306, it is determined if there is an available cache line in the associated set. If there is, then at block 308, data is written to that line. At block 310, it is determined if duplication is complete (i.e., if a given implementation is to write data to two cache lines, then duplication is complete when two cache lines are written to).

[0044] If duplication is not complete, then the method repeats at block 306. If no available cache line exists, then an available cache line must be found through a replacement policy (to be discussed), and when that line is found, data is written to that line back at block 308. If duplication is complete, then the method ends at block 314.

[0045] The method is not restricted to the order described. The method could be performed in a different order. For example, It could first be determined if all available cache lines exist before data is written, rather than writing data to an available cache line as it is found.

Write-Backs

[0046] Write-backs occur to save cache data to main memory. FIG. 4 is a flowchart illustrating a write-back operation within general embodiments of the invention. The method starts at block 400 and continues to block 402 where a first dirty cache line is read for writing back to memory. It is determined if the cache line is corrupt at block 404.

[0047] Cache line is not corrupt: If the line is not corrupt, then the cache line is written back to a corresponding memory location at block 414, as

determined by the tag address of the cache line. At block 418, the cache line is marked available.

[0048] At blocks 408 and 410, an associated cache line corresponding to the current memory location is searched for. A current memory location comprises a memory location that has not been written to yet, where the memory location corresponds to a first read dirty line in an associated set. The current memory location gets reset when a memory location gets written to. Thus, if the first dirty cache line corresponds to a memory location A, memory location A is set as the current memory location. If an associated dirty cache line corresponds to a memory location B, then the associated cache line does not correspond to the current memory location. If an associated dirty cache line corresponds to a memory location A, then it corresponds to the current memory location.

[0049] If the associated cache line corresponds to the current memory location 410 and the current memory location has already been written to 412, then the associated cache line is marked invalid at block 406. This process is repeated until there are no more associated cache lines.

[0050] If the associated cache line corresponds to the current memory location 410 and the current memory location has not been written to 412, then at block 414, the data is read, and the method repeats at block 404 to determine if the data is corrupt.

[0051] If the associated cache line does not correspond to the current memory location 410, then another associated cache line is determined at block 408 until a cache line corresponding to the current memory location is found.

[0052] Cache line is corrupt: If the cache line is corrupt at block 404, then at block 406, the line is marked invalid, and available for use. At block 408, it is determined if an associated cache line exists, and at block 410 it is determined if the associated cache line corresponds to the current memory location. If it does, and if the memory location has not been written to, then the line is read at block 414 to determine if it is corrupt at block 404. If it is not corrupt, it is written to the current memory location at block 416, and the line is marked available at block 418.

[0053] If the current memory location has been written to, then associated cache lines are found and marked invalid. (However, even once a memory location has been written back to, corresponding cache lines may still be read to achieve a high confidence level that the data being written back is valid. Using this approach, the lines may be marked accordingly - i.e., if a line is not corrupt, it can be marked available - clean or dirty - and if a line is corrupt, it is marked invalid. While either implementation is within the scope of the invention, described embodiments use the former of these two implementations.) When no more associated cache lines exist, the method ends at block 420.

Replacement Policy

[0054] Write-backs may occur for any number of reasons as dictated by a

replacement policy implemented for a given cache. A replacement policy is the policy that is used to identify cache lines to replace. A replacement policy may dictate that clean lines are replaced first; dirty lines are replaced first; all lines are replaced; younger lines are replaced first; or least recently used (LRU) lines are replaced first, for example.

[0055] Write-backs can occur, for example, on an as-needed basis when any number of dirty cache lines are freed up for a certain purpose; or for a context switch or a cache flush where all cache lines, clean and dirty, are written back, for example

[0056] Thus, in a four way set-associative cache, each of the four lines in an associated set is read. Generally, if data on a cache line is not corrupt, it is written to memory and marked available, and if it is corrupt, it is not written to memory and marked invalid. The determination of whether an available line is marked clean or invalid is implementation and/or situation dependent.

[0057] If the line is a dirty line and the data is corrupt, it is marked invalid, and a duplicate line is searched for. This process is repeated until no more duplicate lines exist, or until a duplicate line having uncorrupt data is found. A duplicate line having uncorrupt data is written back and then marked available.

Conclusion

[0058] A mechanism has been described for writing data to and from a cache for cost-efficiently protecting dirty data. The mechanism can be implemented with little modification to existing cache architectures, and minimal

modification to existing core processors. Error correction is extended to cover failures over a large number of bits without full duplication or the use of logically complex schemes such as the RAID approach.

[0059] Furthermore, the mechanism is congruent with existing methods of data recovery. For example, in an implementation where both dirty cache lines are read to achieve a high confidence of valid data, if the second cache line is corrupt, it can be scrubbed using the uncorrupt data from the first cache line. As another example, if one of the cache lines becomes permanently non-functional, the capability to run the cache in degraded mode means that the part of the cache containing the non-functional cache line may be ignored with the advantage that the first cache line is not functional, and may contains good data.

[0060] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.